SORBONNE UNIVERSITY

COMPUTER SCIENCE

Image (IMA)

Deep Learning Practical Work 2-de

Candidate:

Caterina Leonelli: 21306668

Luisa Neubauer: 21228141

Academic Year
2023–2024

Contents

Introduction

Generative adversarial networks [1] are models that generate data samples given the statistical distribution of the data. This is done by combining two different networks that have two different purposes: the **generator**, which has the goal to take a random normal distribution z, and outputs a generated sample $G(z)$ that is close to the original distribution; the **discriminator**, that evaluates the output generated by the generator. The evaluation $D(G(z))$ by the discriminator is a value between 0 and 1 where 0 means that the generated sample is judged as fake, while 1 means that is judged as real. Each of the two networks has to learn their job simultaneously: the generator learns to output a random distribution from the input, while the discriminator is fed with a true image x and learns its distribution in order to compute $D(x)$ confirming if x is a true image or not. After this, the random distribution of the generator is evaluated by the discriminator, which will evaluate if the generated distribution is similar or not to the real distribution. The final goal of the jointly two-party system is to train both networks in a way that leads the generator to produce fake distributions as similar as possible to the original ones, but that are unseen and original at each new generation. To do so, the original loss function for the training, defined in 1, searches for a Nash equilibrium that is a minimum for the generator and a maximum for the discriminator, in a so called min-max game.

$$\min_G \max_D V(D,G) = E_x[logD(x)] + E_z[log(1 - D(G(z)))] \tag{1}$$

Where:

- $log(D(x))$ is the cross-entropy between $[1 \quad 0]^T$ and $[D(x) \quad 1 - D(x)]^T$;
- $log(1 - D(G(z)))$ is the cross-entropy between $[0 \quad 1]^T$ and $[D(G(z)) \quad 1 - D(G(z))]^T$.

Practically speaking, $log(D(x))$ and $log(1 - D(G(z)))$ measure the probability that the discriminator is rightly classifying real and fake images, respectively.

Throughout the years researchers found many adversarial loss in order to solve two different basic problems, that arises right from the Nash equilibrium goal [2]: the **vanishing gradient descent** problem and the **mode collapse** problem. While the former arises when the derivative of the loss function with respect to the current weight in each iteration of training is too small that the update to the original weights is not sufficient enough, the latter arises during training when the generator produces images that successfully pass the discriminator's evaluation and so consequently, the generator tends to replicate these images, leading to a lack of originality and resulting in a reduced diversity of generated data samples. An additional relevant problem of GANs is high computational cost. This is due to the fact that two networks has to be trained simultaneously and that the convergence to the Nash equilibrium is not guaranteed.

In this practical work, we will focus on generative models that traditionally model the joint probability P(X, Y) and thus the likelihood of data X. In particular, we will focus on Generative Adversarial Networks (GANs).

Generative Adversarial Networks

1. Interpret the equations (6) and (7). What would happen if we only used one of the two?

If we use only the loss of the equation (6) than the discriminator will not improve, therefore the generator will easily fool the discriminator and generate bad looking images. If we only use equation (7) than the generator will not improve, therefore it will not pass the discriminator check.

2. Ideally, what should the generator G transform the distribution P (z) to?

Ideally the the generator should sample new images that can fool the discriminator check and that are similar to the ones in the train set but also original.

3. Remark that the equation (6) is not directly derived from the equation 5. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true" equation be here ?

Normally the loss function is this:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}^* \in \mathcal{D}\text{ata}} [\log D(\mathbf{x}^*)] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))]$$

so for the G the important part would be:

$$min_G = E_{z \sim P(z)} log(1 - D(G(z)))$$

The equation used by the authors is used in order to avoid the vanishing gradient problem: during the optimization process if we calculate the gradient of this part of the loss it would look like this:

$$D[log(1 - D(G(z)))] = D[\frac{1}{(1-\sigma)}\sigma] = \frac{\sigma(1-\sigma)}{(1-\sigma)} = \sigma$$

And at the beginning this would be closer to zero.

Instead if we rewrite this loss as:

$$max_G = log(D(G(z)))$$

then the derivation would be:

$$D[log(D(G(z)))] = \frac{1}{(\sigma)}\sigma(1-\sigma) = 1 - \sigma$$

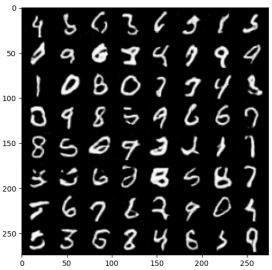
which in the beginning would be going to 1

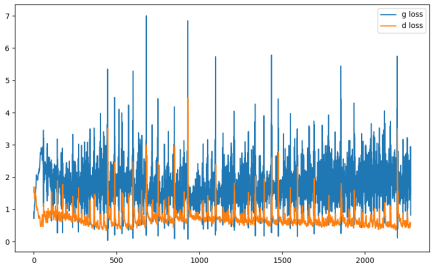
4. Comment on the training of of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.

We train our network with the given parameters, and we obtain images that are quite good and diverse. The loss curves of the generator and discriminator are not stable because the two networks are adversaries. When the generator's loss decreases, the discriminator's loss increases, and vice versa.

This is because they are in a continual adversarial competition: when one network tries to reduce its loss, the other network responds by attempting to increase it, creating a dynamic and fluctuating relationship.

ngf=32 →Channel size before the last layer in Generator
ndf=32 →Channel size in Discriminator
weight_init="custom" →Weight initialization type
loss_type=default →Type of training loss for the generator
lr_d=0.0002 →Learning rate for the discriminator
lr_g=0.0005 →Learning rate for the generator
beta1=0.5 →Beta1 for Adam optimizer
epochs=5 →Number of training epochs
nz=100 →Latent size
batch_size=12 →Batch size
nchannels=1 →Number of channels for inputs of Discriminator

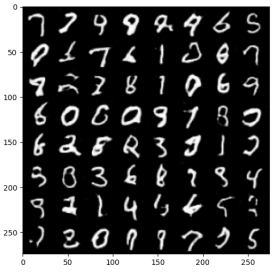


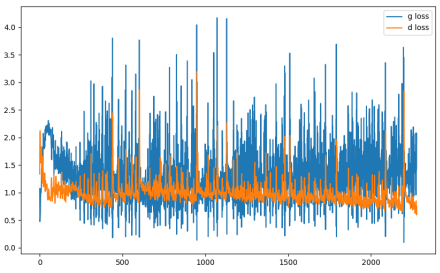


5. Comment on the diverse experiences that you have performed with the suggestions above. In particular, comment on the stability on training, the losses, the diversity of generated images, etc.

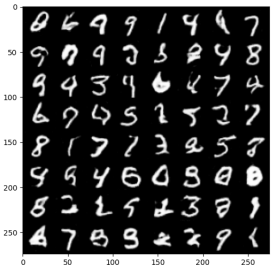
Modify ngf or ndf. In particular, reduce or increase one of the two significantly.

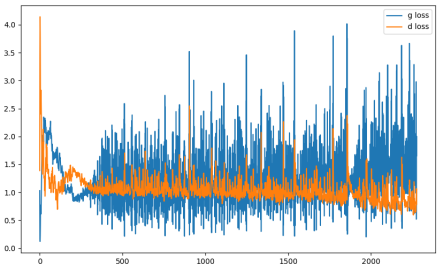
Starting with ngf, if we increase it, we will not obtain significant differences between the baseline. We can see that the discriminator perform a little bit worse that the generator when increasing ndf and also the training seems more stable.
ngf=64



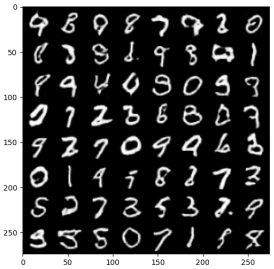


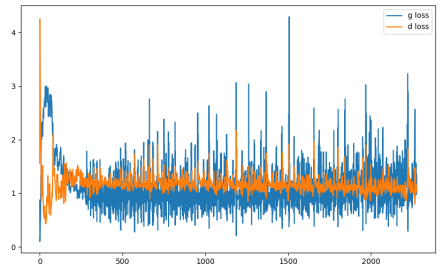
ngf=128



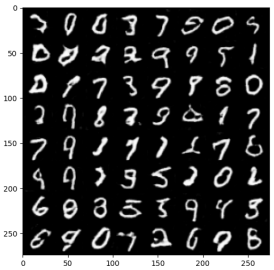


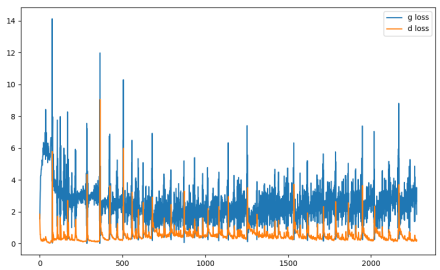
ngf=256



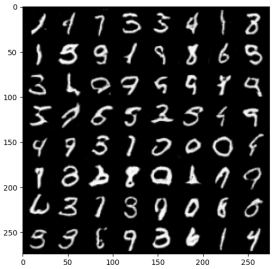


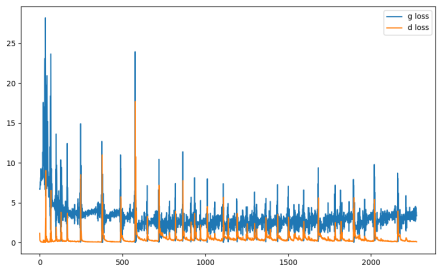
While, increasing the ndf will produce different results: at ndf=256 the performances drop significantly. We can see that the generator loss perform poorer and poorer each time we increase ndf. One explanation could be that the discriminator becomes too powerful compared to the generator (i.e., it has a higher capacity). Moreover if the discriminator is too complex, the gradients during backpropagation may become very small (vanish), making it challenging for the generator to learn and update its parameters effectively.
ndf=64



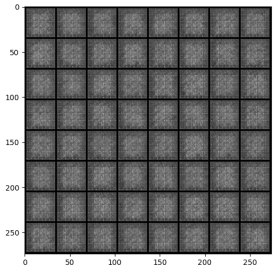


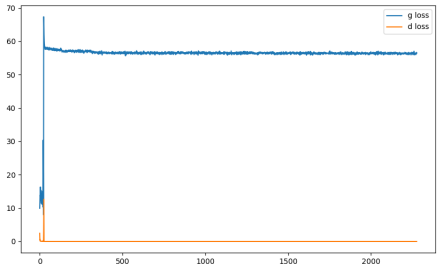
ndf=128





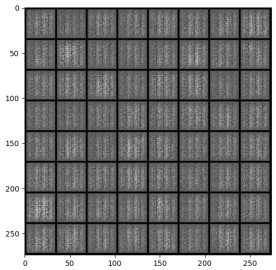
ndf=256

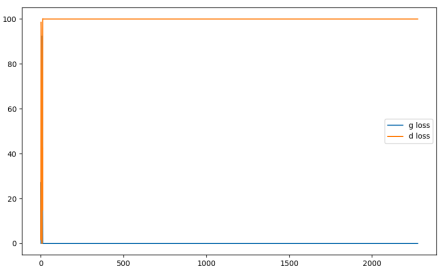




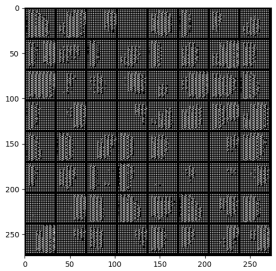
Change the learning rate of one or both models

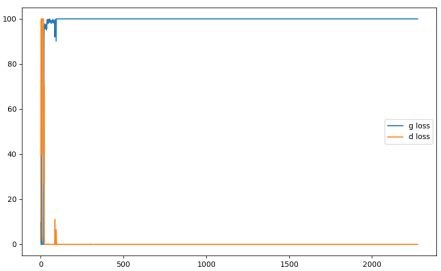
Increasing the learning rate of one or both the models will produce really bad results and an unstable training:
lr_d=0.1



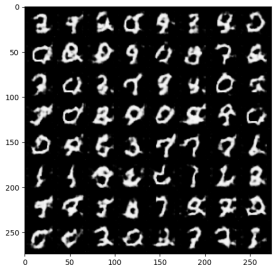


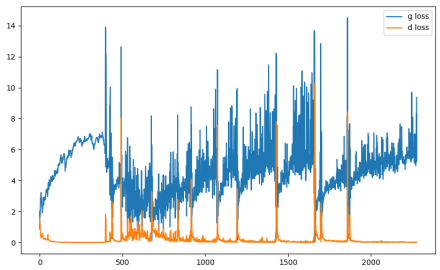
lr_d=0.1, lr_g=0.1



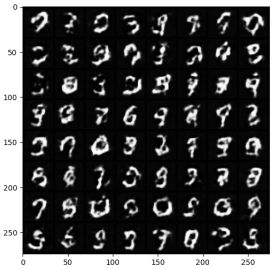


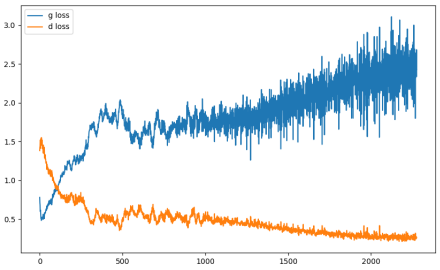
lr_g=0.1



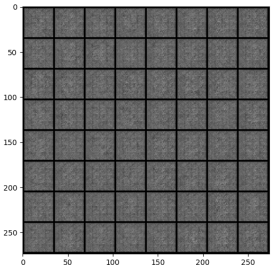


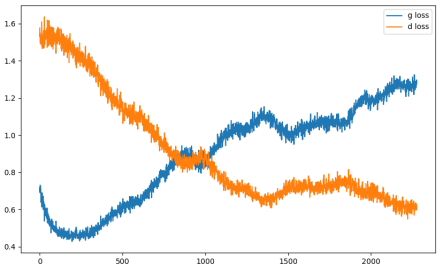
Also decreasing the learning rate of both models will produce worse results from baseline. In fact, we can see that the generator loss is increasing while the discriminator loss is decreasing and this is the opposite outcome that we would like to obtain.
lr_d=0.00002, lr_g=0.00005





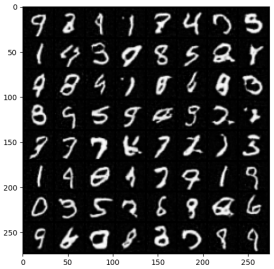
lr_d=0.000002, lr_g=0.000005

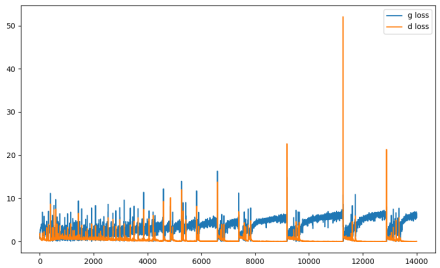




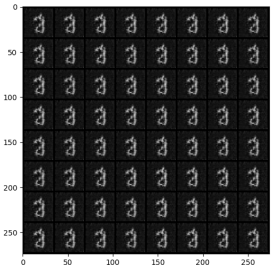
Learn for longer (ex : 30 epochs) even if it seems that the model already generates correct images

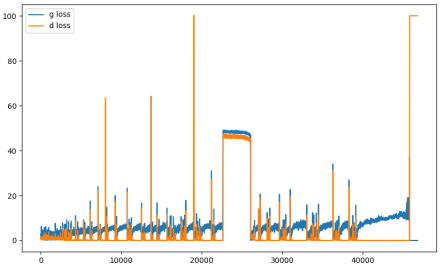
After step 3000 we can see that the training gets more unstable: the two losses experience periods of diverging moment and then a spike. In our opinion, this could be the effect of an exploding gradient phenomenon. Also the generator collapses to producing a limited set of samples, ignoring the diversity present in the training data. This is the "mode collapse problem" resulting in generated samples that lack variety. We can see an extreme effect of this problem in the test with epochs=100.
epochs=30





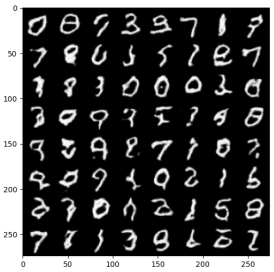
epochs=100

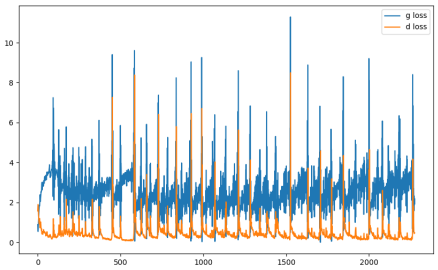




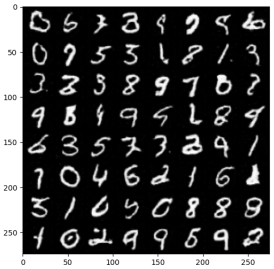
Reduce or increase significantly nz (ex : nz = 10 ou 1000)

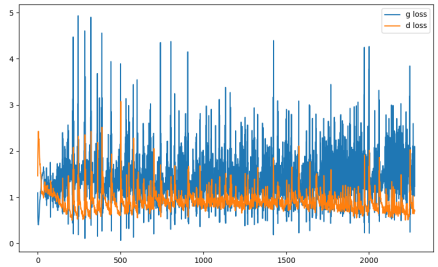
Increasing the nz significantly will make the training more unstable: a very high-dimensional latent space might lead to overparameterization, where the model has more parameters than necessary. Overparameterization can make the training process slower and potentially lead to overfitting. Also a larger latent space requires more diverse and plentiful data to effectively capture the underlying data distribution. If the dataset is not sufficiently large or diverse, the model might struggle to learn meaningful representations.
nz=100





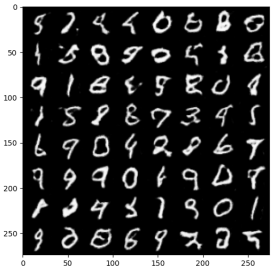
If we decrease it, we obtain good results compared to the baseline. A lower-dimensional latent space simplifies the model, making it computationally more efficient and reducing the risk of overfitting. With fewer parameters, the model might generalize better to unseen data.
nz=1000

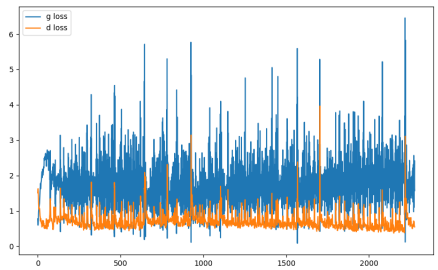




Replace the custom weight initialization with pytorch's default initialization

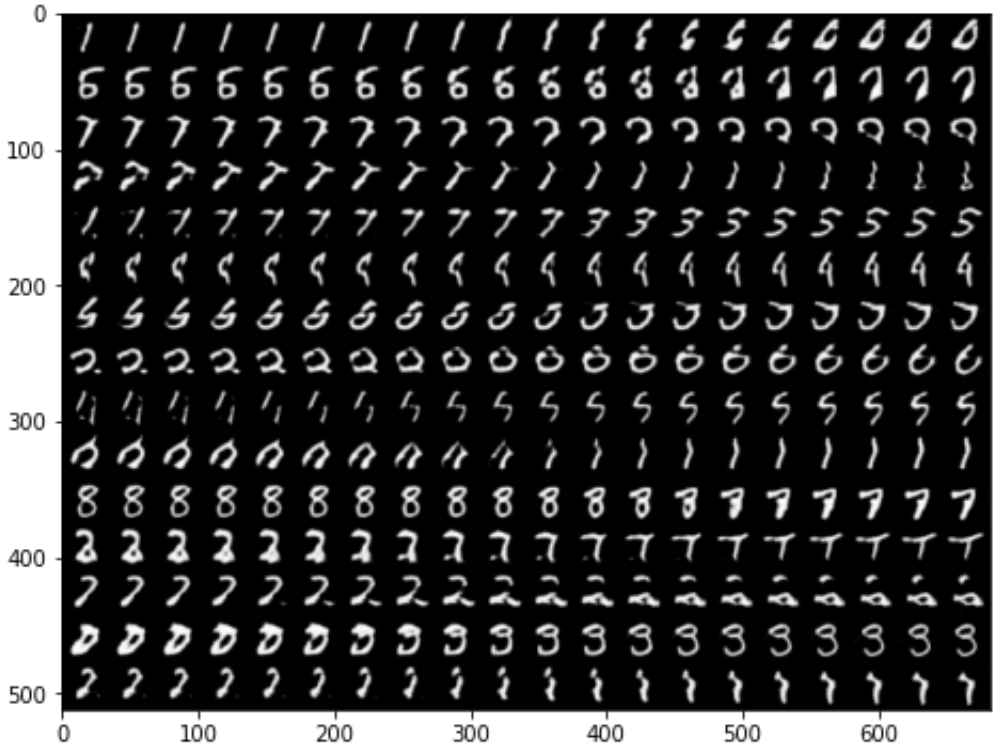
Training with the default initialization will make us obtain similar results as the baseline.
init_weight="pytorch"





Using a learned GAN, take 2 noise vectors z1 and z2 and generate the images corresponding to several linear interpolations $\hat{I} \pm z_1 + (1\hat{a}\hat{I}\pm)z_2, \hat{I} \pm \hat{a}[0,1]$**.**

This technique serves to explore the variety of results that a GAN can achieve. As α varies from 0 to 1, the linear interpolation results in smooth transitions between the generated images.


0
100
200
300
400
500
0
100
200
300
400
500
600

Conditional Generative Adversarial Networks

6. Comment on your experiences with the conditional DCGAN.

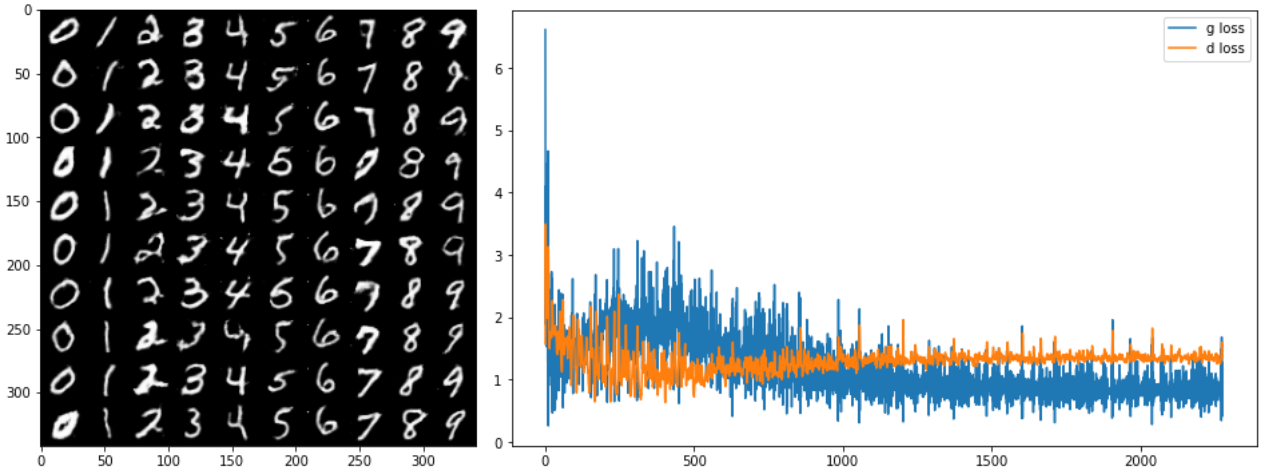
We train a cDCGAN on MNIST using the image class as conditioning (with a one-hot vector). The training of a conditioned GAN is more stable than that of a traditional GAN. The images generated in the middle of the training are more noisy; they exhibit numerous white points regardless of the input vector, and this phenomenon is accompanied by higher losses. At the end of our experiment, the generated images are of good quality, and the networks seem to have stabilized.

7. Could we remove the vector y from the input of the discriminator (so having cD(x) instead of cD(x, y)) ?

It is important that the discriminator and generator have access to the y label to generate a desired digit. If we remove the label y from the discriminator inputs, the model will not learn to generate under a condition. The generated numbers will be random like a normal GAN. Additionally, experiments have demonstrated that this alteration typically leads to a decrease in the model's performance. The inclusion of y adds valuable contextual information that helps the discriminator make more accurate decisions. Therefore, while simplifying the model by removing y is an option, it's important to consider the potential trade-offs in terms of effectiveness and accuracy.

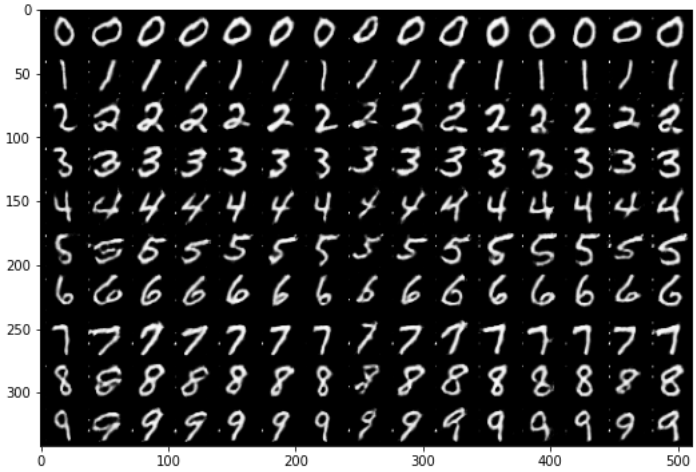
8. Was your training more or less successful than the unconditional case ? Why?

Our training was more successful in the conditional case compared to the unconditional one, since the conditional variable helps refining the learning process. The training seems to be smoother and more effective, with less high spikes compared to the baseline of the unconditional GAN. Also, in the end the discriminator loss is going upwards, while the generator loss is still decreasing and this is the behavior that we want to achieve.



9. Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here?

As seen from the generation, the latent vector z only changes the style of the generated image. Since we have given the condition and it learns according to that, the output image is from the same category/class. This can be seen from the image.



Bibliography

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML].

[2] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, *A review on generative adversarial networks: Algorithms, theory, and applications*, 2020. arXiv: 2001.06937 [cs.LG].